REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
31-10-2013	Final	01-08-2009 - 31-10-2013

4. TITLE AND SUBTITLE

Final Report for Scalable and Accurate SMT-based Model Checking of Data Flow Systems

5a. CONTRACT NUMBER

FA9550-09-1-0596

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Clark Barrett and Morgan Deters

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

New York University
70 Washington Square S
New York, NY 10012-1019

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

USAF/AFRL DUNS 143574726
AF OFFICE OF SCIENTIFIC RESEARCH
875 N RANDOLPH ST. ROOM 3112
ARLINGTON, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In this project, we developed CVC4, a new SMT solver for use in verification and other applications, especially in the KIND model checker developed collaboratively at U Iowa. CVC4 was developed from scratch into an award-winning 200K+ LOC solver extensively used and cited by academic and industry users around the world. CVC4 is competitive with the very best SMT solvers and outperforms all other SMT solvers on certain classes of problems.

15. SUBJECT TERMS

Verification, model-checking, SMT, satisfiability modulo theories

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Clark Barrett
U	U	U		20	19b. TELEPHONE NUMBER *(Include area code)* 212-300-6909

Reset

1 Introduction

Formal verification is crucial to ensuring hardware and software quality. Microchip manufacturers regularly apply verification to ensure the proper behavior of their products, and developers designing safety-critical software employ verification to provide assurance that the software meets safety constraints.

One of the key technologies supporting formal verification of hardware and software is that of Satisfiability Modulo Theories (SMT). SMT solvers operate by solving formulas in logic. These formulas capture the behaviors of hardware and software, and can be written to test the possibility of unwanted behavior. A solution provided by an SMT solver affirms the presence of unwanted behavior, or, conversely, proves its absence.

SMT formulas are Boolean combinations of theory constraints. Theories of interest in SMT typically target verification applications, and thus they are geared toward the accurate specification of software, hardware, memory states, and related data. Theories commonly supported by SMT solvers are:

- Theories of arithmetic over integers, reals, and mixed reals and integers. Virtually all software contains arithmetic; arithmetical constraints (e.g., the value of variable x is always less than that of variable y) can be represented in this theory.

- A theory of inductive datatypes. Modeling software datatypes can be done directly in this theory.

- A theory of arrays. Software that uses arrays can be modeled with constraints in this theory, as can computer memories (which are, essentially, large arrays of values).

- A theory of fixed-width bitvectors. For modeling hardware circuitry, constraints over bitvectors are often needed. In software, "integers" are not true mathematical integers, but bitvectors; this theory is thus indispensable in the accurate modeling of both hardware and software.

- A theory of uninterpreted functions. First-order theories not supported directly by a solver can be axiomatized with the help of functions, and parts of the analysis that don't require precision can be abstracted away by using a function, simplifying the formula.

Additional theories are sometimes supported, such as a theory of strings, a theory of sets, or a theory of floating-point. All of these are *first-order theories* supporting quantification over relevant domains. For example, one can write an SMT formula representing the statement that "all integers satisfy a certain property." It is well-known that for many theories, such quantified statements cannot always be answered affirmatively or negatively; the satisfiability problem for such theories is said to be *undecidable,* and SMT solvers must sometimes yield an answer of "unknown" when facing such problems.

The challenges in constructing SMT solvers lie in efficiently solving problems of practical interest. To that end, new heuristics are developed to solve formulas more quickly; new theories and restrictions of theories are identified that are useful for verification and other applications; and new, practical decision procedures are implemented for these theories and their restrictions.

Over the four years of this AFOSR project, a new, innovative SMT solver named CVC4 has been designed, developed, and released. CVC4, the *Cooperating Validity Checker* version 4, incorporates many new advancements over its predecessors and over other available SMT solvers. Significant theoretical and practical advances were made to make CVC4 applicable to problems of interest. In particular, all of the above-mentioned theories were supported by CVC4 or in development at the conclusion of this project. Further, SMT challenges were addressed by implementing novel decision procedures for theories of interest, and a new combination framework was developed for deciding formulas over combined theories (for example, arrays of bitvectors combines the theory of arrays and the theory of bitvectors). Finally, advances were made to identify cases where CVC4 can confidently answer affirmatively or negatively in cases where, due to quantification, other solvers yield an answer of "unknown."

1.1 Motivation and impact

Great advances have been made in automated reasoning by research into SMT. The field of SMT itself has gained significant attention since 2005 (see Figure 1). Within this hot research area, many SMT solvers have been developed; a few examples of modern, popular SMT solvers are Barcelogic [6], MathSat [7] OpenSMT [8], Yices [10], and Z3 [9]. These solvers support somewhat different feature sets and performance profiles.

CVC4 was developed at an opportune time: its design took advantage of considerable research into SMT in the years preceding the effort. Further,

the development team, with decades of combined experience in these solvers, was well-situated to design a new tool. Novel research was conducted during the design phase to determine the best course of action, and many novel products of research are found in CVC4. CVC4 has consistently performed well in competitions against other, similarly-capable solvers; more details are given in Section 3. In the first year after its initial public release, over 50 articles (as counted by Google Scholar) have referenced the project.

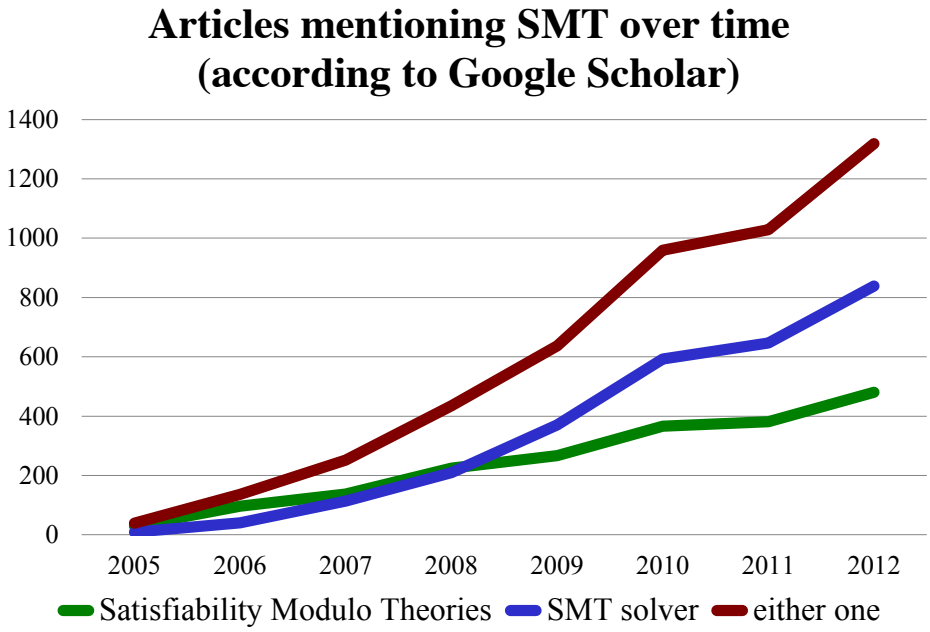

Figure 1: SMT citations

1.2 SMT-LIB

The SMT-LIB initiative was formed in 2003 to promote a standard for SMT solvers and provide a large, common library of benchmarks for SMT solvers. A significant update to the standard was made in 2010, and new benchmarks are incorporated regularly. CVC4 was developed to be fully compliant with both versions this community standard:

- CVC4 was built to support (nearly) all of the theories and combinations

of theories prescribed by SMT-LIB. Plans were made to address deficiencies in non-linear arithmetic to extend this support to all SMT-LIB theories.

- CVC4 was designed to support the full input language of SMT-LIB. CVC4 supports a number of convenient extensions to the standard, but also provides a "compliance mode" that disables these features and strictly follows the word of the standard.

- During development, CVC4 was continuously evaluated on the SMT-LIB benchmark library. CVC4 showed good overall performance on these benchmarks, and it showed superior performance on some important classes of benchmarks; Section 3 contains experimental highlights.

The above contributions furthered the goals of SMT-LIB during the course of this project, providing a new, performant SMT solver that was largely interoperable with other existing tools.

1.3 History

The Cooperating Validity Checker series has a long history. The Stanford Validity Checker (SVC) [4] came first, incorporating theories and its own SAT solver. Its successor, the Cooperating Validity Checker (CVC) [19], had a more optimized internal design, produced proofs, used the Chaff [16] SAT solver, and featured a number of usability enhancements. Its name comes from the *cooperative* nature of decision procedures in Nelson-Oppen theory combination [17], which share amongst each other equalities between shared terms. CVC Lite [1], first made available in 2003, was a rewrite of CVC that attempted to make CVC more flexible (hence the "lite") while extending the feature set: CVC Lite supported quantifiers where its predecessors did not. CVC3 [5] was a major overhaul of portions of CVC Lite: it added better decision procedure implementations, added support for using MiniSat [11] in the core, and had generally better performance.

CVC4 is the new version, the fifth generation of this validity checker line that is now celebrating seventeen years of heritage. It represents a complete re-evaluation of the core architecture to be both performant and to serve as a cutting-edge research vehicle for the next several years. Rather than taking CVC3 and redesigning problem parts, a clean-room approach was taken, starting from scratch. Before using any designs from CVC3, they

were thoroughly scrutinized, vetted, and updated. The components of CVC4 bear only a superficial resemblance, if any, to their correspondents in CVC3. However, CVC4 is fundamentally similar to CVC3 and many other modern SMT solvers: it is a DPLL(T) solver [12], with a SAT solver at its core and a delegation path to different decision procedure implementations, each in charge of solving formulas in some background theory. The re-evaluation and ground-up rewrite was necessitated by the performance characteristics of CVC3. CVC3 had many useful features, but some core aspects of the design led to high memory use, and the use of heavyweight computation (where more nimble engineering approaches could suffice) made CVC3 a much slower prover than other tools. As these designs were central to CVC3, a new version was preferable to a selective re-engineering.

2 The CVC4 System

CVC4 is the latest version of the Cooperating Validity Checker. A joint project of New York University and The University of Iowa, CVC4 aims to support the useful feature set of CVC3 and SMT-LIBv2 while optimizing the design of the core system architecture and decision procedures to take advantage of recent engineering and algorithmic advances. CVC4 represents a completely new code base; it is a from-scratch rewrite of CVC3, and many subsystems were completely redesigned.

2.1 Key features

Key features of CVC4 are previewed here, and described more fully in the sections that follow.

Support for a variety of theories. CVC4 supports functions, linear arithmetic, bit-vectors, arrays, and inductive datatypes. Additional theories are currently in development.

Quantifiers. CVC4 supports quantification through heuristic instantiation and includes a novel finite model finder.

Proofs and counterexamples. CVC4 can emit partial mathematical proofs of proven formulas. For formulas that can be falsified, counterexamples are produced. For applications like verification, these features are crucial. A proof can be machine-checked for correctness; CVC4 itself need not be trusted, as

the proof is sufficient to ensure correctness of the analysis. A counterexample provides the ability to see exactly where the unwanted behavior is exhibited (i.e., where the bug is).

Modularity. A modular design and well-structured internal interfaces make it much easier to extend CVC4 than for other projects of its size and scope.

Cross-language and cross-platform support. CVC4s API can be accessed from C, C++, Java, and OCaml, and provisions have been made to support other languages. CVC4 can be compiled and run on various flavors of Linux, Mac OS, and Windows. An automated build system regularly builds and tests many different configurations on these platforms to ensure quality.

Community standards compliance. In addition to CVC4s library API, CVC4 supports standard textual input languages, such as SMT-LIB and TPTP.

2.2 Architecture

CVC4 is organized around a central core of *engines* (see Figure 2):

- The SMT Engine serves as the main outside interface point to the solver. The SMT Engine has public functions to push and pop solving contexts, manipulate a set of currently active assumptions, and check the validity of a formula, as well as functions to request proofs and generate models. This engine is responsible for setting up and maintaining all user-related state.

- The Prop Engine manages the propositional solver at the core of CVC4. This, in principle, allows different SAT solvers to be plugged into CVC4.

- The Theory Engine serves as an "owner" of all decision procedure implementations. As is common in the research field, these implementations are referred to as *theories* and all are derived from the base class Theory.

CVC4's Theory class serves as the basis for decision procedure implementations. A decision procedure extends this class to check consistency of conjunctions of theory constraints, participate in preprocessing, and establish a canonical form for theory constraints, among other purposes.

CVC4 also incorporates *managers* in charge of managing subsystems:

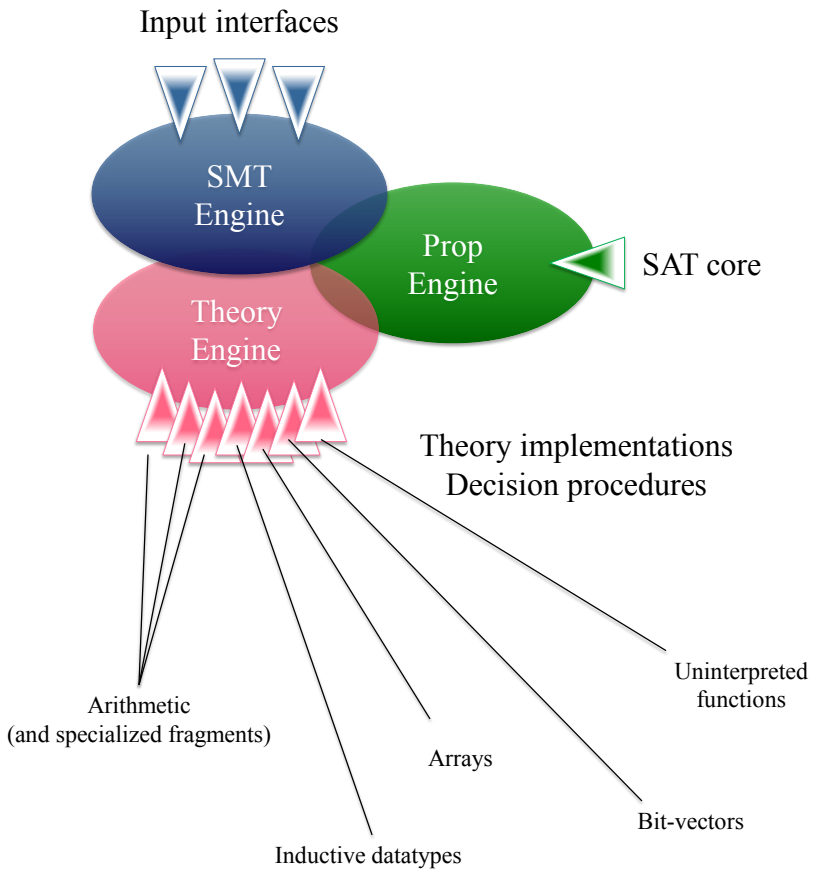

Figure 2: Architecture of CVC4.

- The Node Manager is one of the busiest parts of CVC4, in charge of the creation and deletion of all expressions ("nodes") in the prover. Node objects are immutable and subject to certain simplifying constraints. Further, Node objects are unique; the creation of an already-extant Node results in a reference to the original. Node data is reference-counted (the Node class itself is just a reference-counted smart pointer to node data) and subject to reclamation by the Node Manager when no longer referenced; for performance reasons, this is done lazily (see below for performance justification).

- The Context Memory Manager is in charge of maintaining a coherent, backtrackable data context for the prover. At its core, it is simply a region memory manager, from which new memory regions can be requested ("pushed") and destroyed ("popped") in LIFO order. These regions contain saved state for a number of heap-allocated objects, and when a pop is requested, these heap objects are "restored" from their backups in the region. This leads to a nice, general mechanism to do backtracking without lots of *ad hoc* implementations in each theory; this is highly useful for rapid prototyping. However, as a general mechanism, it must be used sparingly; it is often beneficial to perform backtracking manually within a theory using a lighter-weight method, to timestamp to indicate when a previously-computed result is stale, or to develop approaches requiring little or no backtracking at all (e.g., tableaux in Simplex).

- The Proof Manager is charged with collecting parts of proofs and stitching them together into a proper proof in the desired output format.

2.3 Expressions

Expressions are considerably more efficient than CVC3's expression representation. In the latest version of CVC3, expressions maintain 14 word-sized data members (plus pointers to child expressions). In CVC4, nodes require 64 bits plus child pointers, a considerable space savings. (In part, this savings results from clever bit-packing. Part is in storing node-related data outside of Node objects when appropriate.)

The expression subsystem of CVC4 has been carefully designed, and runtime profiling data was analyzed to ensure its performance was reasonable. On stress tests, it beat CVC3's expression subsystem considerably [2].

2.4 Theories

CVC4 incorporates newly-designed and implemented decision procedures for its theory of uninterpreted functions, its theory of arithmetic, of arrays, of bitvectors, and of inductive datatypes, based on modern approaches described in the literature. Performance generally was determined far superior to CVC3's, and competitive with other solvers on a broad range of benchmarks. (See Section 3.)

2.5 Proofs

CVC4's proof system was designed to support LFSC proofs [18], and was also designed to have *absolutely zero footprint* in memory and time when proofs are turned off at compile-time.

2.6 Library API

As CVC4 is meant to be used via a library API, there's a clear division between the public, outward-facing interface, and the private, inward-facing one. This decision was made to provide a number of benefits to its users. First, a CVC4 installation doesn't require numerous, private interfaces to be installed on a user's machine. Outward-facing symbols (functions, classes, globals, etc.) are marked for dynamic library export and inward-facing ones are not. This can speed dynamic linking and library-internal function calls, and ensures only functions intended to be public can be invoked.

Additionally, the approach ensured that public dependences cannot become tangled with private ones. This clear separation also made it easy to generate documentation only for the public API, and also to flag undocumented parts of the public interface.

Further, in some cases, different classes are used in public and private contexts for the same entity. Expressions are a "public" view of nodes, which are slightly heavier-weight but also safer, carrying a reference back to their owning ExprManager. This permits a client program to create two distinct copies of CVC4 in memory safely, even sharing some expressions between them, and the library can properly ensure that memory is correctly used and reclaimed.

Finally, CVC4's own parser and main command-line tool were designed to link against the CVC4 library in the same way that any other application

would. This helped to ensure that the library API was complete.

2.7 Backward compatibility

CVC4 provides a new, streamlined API for accessing the core routines of the system. However, CVC4 also exports an API identical to that of CVC3, with the aim of achieving source-level compatibility with CVC3. Users of the previous version therefore can benefit from the improvements in CVC4 without re-designing and re-implementing their tools.

Further, the native input language of CVC4 is backward-compatible with CVC3's, with minor exceptions. This has allowed CVC4 to be used as a drop-in replacement in many places where CVC3 was used.

2.8 Theory modularity

Theory objects are designed in CVC4 to be highly *modular:* they do not employ global state, nor do they make any other assumptions that would inhibit their functioning as a client to another decision procedure. In this way, one Theory can instantiate and send subqueries to a completely subservient client Theory without interfering with the main solver flow.

2.9 Support for concurrency

CVC4's infrastructure has been designed to make the transition to multiprocessor and multicore hardware easy, and a lemma-sharing portfolio version of CVC4 is maintained. CVC4 shows promise as a good vehicle for other research ideas in this area. In part, the modularity of theories (above) was geared toward this—the absence of global state and the immutability of expression objects clearly makes it easier to parallelize operations. Similarly, the Theory API specifically includes the notion of *interruptibility*, so that an expensive operation (e.g., theory propagation) can be interrupted if work in another thread makes it irrelevant.

2.10 Extensive documentation

A strong goal of the CVC4 development project has been to make it relatively easy for a newcomer to extend. To realize this goal, extensive documentation was developed, including:

- a developers' wiki that evolves along with the design of the project, serving as a log of design history and current best practices for the team;
- meeting minutes and electronic whiteboard notes for all developers' meetings;
- a developers' guide with coding practices, code repository policies, design and internal API notes, and "how-to"-like sections to aid in development.

A website for CVC4 is maintained at `http://cvc4.cs.nyu.edu/`.

2.11 Project summary

CVC4 aims to follow in CVC3's footsteps as an open-source theorem prover supporting this wide array of background theories. CVC3 supports all of the background theories defined by the SMT-LIB initiative, and provides proofs and counterexamples upon request; CVC4 aims for full compliance with the new SMT-LIB version 2 command language and backward compatibility with the CVC presentation language.

In this way, CVC4 is a drop-in replacement for CVC3, with a cleaner and more consistent library API, a more modular, flexible core, a far smaller memory footprint, and better performance characteristics.

The increased performance of CVC4's (over CVC3's) expression subsystem was demonstrated in [2]; CVC4's solving apparatus also performs better than CVC3's. In the 2010 annual SMT competition [3], both solvers competed in the QF_LRA division. CVC4 solved more than twice the benchmarks CVC3 did, and for the benchmarks they both solved, CVC4 was almost always faster. In the most recent competition (in 2012), a prerelease version of CVC4 entered more divisions than in the past, and performed even better than it had before, handily placing above CVC3; a discussion is found in Section 3.

Two main goals for the CVC4 project were to provide a better-performing implementation of CVC3's feature set, while focusing on flexibility so that it can function as a research vehicle for years to come. This goal was realized for the features that CVC4 currently supports. The second goal was met as well: a number of internal, complicated, non-intuitive assumptions on which CVC3 rests have been removed in the CVC4 redesign. The component interactions

and the data structures were greatly simplified, which made it far easier to document the internals, train new developers, and add support for new features.

3 Experimental Results

CVC4 exhibits good performance compared to other SMT solvers. This section provides some experimental highlights.

3.1 SMT-COMP

Between 2005 and 2012, an annual SMT solver competition was run. A prerelease version of CVC4 performed quite well in SMT-COMP 2012, winning the QF_UFLRA division, which combined the theory of uninterpreted functions and the theory of (linear) arithmetic over reals. In QF_UFLIA, which combines uninterpreted functions and (linear) arithmetic over integers, CVC4 performed well, but as it was a prerelease version, it contained a bug that kept it from placing. A trivially-fixed version, immediately submitted to the competition panel, achieved second place (unofficially) in this category. Other divisions (notably the divisions with quantifiers) did not get enough entrants to run officially; however, the results showed CVC4 with significant improvement over CVC3. This was the case in all competition divisions where CVC3 and CVC4 both competed.

Full results for the annual SMT-COMPs are archived at `http://www.smtcomp.org`.

3.2 Theorem-proving

CVC4 was also entered the CADE Automated Theorem Proving System Competition (CASC 24) in 2013. This research community is somewhat different than SMT's research community, and different standards have been developed for these systems. Nonetheless, as an outsider, CVC4 performed well in the two divisions of this competition that it entered—*first-order formulas* (FOF), in which participants prove the validity of first-order logic statements, and *first-order non-theorems* (FNT), in which participants refute a formula's validity by providing a counterexample.

The performance of CVC4 was judged to be quite good in the competition in both divisions. In FOF, CVC4 performed in the top half of the entrants, though it didn't place due to its inability at the time to output full proofs of these formulas. In FNT, CVC4 had a strong showing, placing third [20].

3.3 SMT-LIB

The SMT-LIB library consists of over 95,000 benchmarks involving arrays, uninterpreted functions, bitvectors, linear and non-linear arithmetic over integers, reals, and mixed integers and reals, as well as many useful combinations of these theories. Many categories of SMT-LIB benchmarks also contain quantifiers. CVC4 has demonstrated very good performance on the SMT-LIB benchmark set generally, and superior performance in some key areas.

Notably, CVC4's linear arithmetic component, based on Simplex and representing four years of devoted effort, solved more problems in the linear real arithmetic (LRA) benchmark set than any other SMT solver it was compared to. Details are found in [15].

3.4 Model-checking

One of the key intentions for CVC4 was to serve as a back-end for the KIND model checker [14], developed at The University of Iowa, which checks invariant properties for Lustre programs. Significant efforts were made to integrate KIND and CVC4 effectively.

KIND has traditionally used the Yices SMT solver, and could also be configured to use CVC3. When using CVC3, KIND was never particularly performant: it was unable to check many properties of interest, properties that it succeeded in checking when paired with Yices. Part of this project aimed to replace Yices as the back-end of KIND, and much progress was made on that front.

The benchmarks in use were the standard set of benchmarks used for KIND performance comparisons [13]. The parallel version of KIND (PKIND) was used, and PKIND+Yices and KIND+CVC4 were compared. They were able to solve the same set of benchmarks: 715 were solved by both PKIND+Yices and PKIND+CVC4, and the remaining ones timed out after 1000 seconds in both cases. As in [13], simple cases were excluded, where both were able to solve the benchmark in less than two seconds. Remaining

were 47 "interesting" benchmarks for the performance comparison. Of these, PKIND+Yices was faster on 18, and PKIND+CVC4 was faster on 29.

4 Papers and Artifacts

CVC4 was designed as a high-performance, industrial-strength SMT solver, to support teams needing such a tool. CVC4 was also intended to be useful within research groups as a vehicle for research in decision procedures and search heuristics. CVC4 was designed specifically from the ground up to be flexible and extensible, and to support rapid prototyping of new research ideas. CVC4 has served as a basis for much of the work of the research team funded by this project, and some 50 articles on Google Scholar have mentioned it within a year of its first public release.

4.1 Documentation and other information

The project developed a website at `http://cvc4.cs.nyu.edu/`. Full documentation of the API, input languages, and features, as well as tutorials and useful examples of key features, are available.

4.2 Source code

CVC4 is open-source and consists of over 200,000 lines of source code. Full source code is available at `http://cvc4.cs.nyu.edu`. The source code repository is open and available at `https://github.com/cvc4/cvc4/`.

4.3 Peer-reviewed publications

Barrett, Conway, Deters, Hadarean, Jovanović, King, and Tinelli. **CVC4.** *Computer Aided Verification (CAV),* 2011.

> This paper described the effort to develop CVC4 between 2009 and 2011. Some preliminary experimental results were described, and the overall architecture of CVC4 was presented.

Jovanović and Barrett. **Sharing is caring: combination of theories.** *Frontiers of Combining Systems (FroCoS),* 2011.

This paper described a new approach for solving the problem of theory combination, making both theoretical and practical contributions to the field. For benchmarks combining arrays and bitvectors, the new approach was demonstrated superior to existing techniques.

Jovanović and Barrett. **Being careful about theory combination.** *Formal Methods in System Design (FMSD),* vol. 42, 2013.

This was a significantly extended version of the FroCoS 2011 paper, adding proofs for all theoretical contributions, providing additional explanations and examples, and simplifying the presentation.

Jovanović, Barrett, and de Moura. **The design and implementation of the model constructing satisfiability calculus.** *Formal Methods in Computer-Aided Design (FMCAD),* 2013.

This paper extended CVC4's core functionality, attempting a new approach to solver design which allows the combination of recently-developed model-based procedures. Encouraging experimental results were reported.

King and Barrett. **Exploring and categorizing error spaces using BMC and SMT.** *Satisfiability Modulo Theories (SMT),* 2011.

This paper reported a case study in verifying a simple version of the Traffic Collision Avoidance System (TCAS), categorizing and reporting problematic states in that system. The paper generalized this experience to a methodology that can be applied in other situations.

King, Barrett, and Dutertre. **Simplex with sum of infeasibilities for SMT.** *Formal Methods in Computer-Aided Design (FMCAD),* 2013.

This paper described a novel decision procedure for linear arithmetic, based on Simplex, that outperforms other approaches on certain difficult benchmark sets. The paper makes contributions of both theoretical and practical significance. The new approach was implemented, and results reported.

Reynolds, Tinelli, Goel, and Krstić. **Finite model finding in SMT.** *Computer Aided Verification (CAV),* 2013.

> This paper described a way to lift a limitation of many SMT solvers that address quantification by heuristic instantiation. A finite model finder for CVC4 was developed and reported in this paper which allows CVC4 to give an answer where it previously could not, and where many SMT solvers cannot.

Reynolds, Tinelli, Goel, Krstić, Deters, and Barrett. **Quantifier instantiation techniques for finite model finding in SMT.** *Automated Deduction (CADE),* 2013.

> This paper extends the effectiveness of the finite model finder described by the CAV 2013 paper (above), by considering and evaluating different techniques for quantifier instantiation. Experimental evidence demonstrates that the approach is practical.

4.4 Technical reports

Technical reports are available from `http://cs.nyu.edu/webapps/content/research/technical_reports`.

Jovanović and Barrett. **Sharing is Caring: Combination of Theories**, TR2011-940, 2011/10

> An extended version of the FroCoS 2011 paper (above), this technical report provided additional material, including proofs and working notes.

4.5 Theses and dissertations

All Ph.D. disserations are available from `http://cs.nyu.edu/web/Research/theses.html`.

Jovanović. **SMT Beyond DPLL(T): A New Approach to Theory Solvers and Theory Combination.** 2012.

This dissertation made contributions in three key areas: deciding linear arithmetic over the integers, deciding non-linear arithmetic over the reals, and deciding combinations of theories. In each case, new approaches were presented, implemented, and evaluated.

Ge. **Solving Quantified First Order Formulas in Satisfiability Modulo Theories.** 2010.

This dissertation proposed novel techniques for solving first-order formulas with quantifiers. Traditional approaches to the problem employ heuristic instantiation. This work studied different heuristics and introduced new ones, and identified situations where heuristic quantifier instantiation can be made complete.

Conway. **Tools and Techniques for the Sound Verification of Low Level Code.** 2010.

This dissertation described CASCADE, a project aimed at the verification of low-level C code. CASCADE was developed to use CVC as a back-end solver.

References

[1] Clark Barrett and Sergey Berezin. CVC Lite: A new implementation of the Cooperating Validity Checker. In Rajeev Alur and Doron A. Peled, editors, *Proceedings of the* 16^{th} *International Conference on Computer Aided Verification (CAV '04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 515–518. Springer-Verlag, July 2004. Boston, Massachusetts.

[2] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *Proceedings of the 23rd international conference on Computer aided verification*, CAV'11, pages 171–177, Berlin, Heidelberg, 2011. Springer-Verlag.

[3] Clark Barrett, Morgan Deters, Albert Oliveras, and Aaron Stump. SMT-COMP 2010: the 2010 edition of the satisfiability modulo theories competition. `http://www.smtcomp.org/2010/`.

[4] Clark Barrett, David Dill, and Jeremy Levitt. Validity checking for combinations of theories with equality. pages 187–201. Springer-Verlag, 1996.

[5] Clark Barrett and Cesare Tinelli. CVC3. In Werner Damm and Holger Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany.

[6] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodrguez-Carbonell, and Albert Rubio. The Barcelogic SMT solver. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 294–298. Springer Berlin / Heidelberg, 2008.

[7] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzn, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT solver. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 299–303. Springer Berlin / Heidelberg, 2008.

[8] Roberto Bruttomesso, Edgar Pek, Natasha Sharygina, and Aliaksei Tsitovich. The OpenSMT solver. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153. Springer Berlin / Heidelberg, 2010.

[9] Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of the Theory and practice of software, 14th international conference on Tools and algorithms for the construction and analysis of systems*, TACAS'08/ETAPS'08, pages 337–340, Berlin, Heidelberg, 2008. Springer-Verlag.

[10] Bruno Dutertre and Leonardo de Moura. The YICES SMT solver. `http://yices.csl.sri.com/tool-paper.pdf`.

[11] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 333–336. Springer Berlin / Heidelberg, 2004.

[12] Harald Ganzinger, George Hagen, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. DPLL(T): fast decision procedures. pages 175–188. Springer, 2004.

[13] George Hagen and Cesare Tinelli. Scaling up the formal verification of lustre programs with smt-based techniques. In *Proceedings of the 2008 International Conference on Formal Methods in Computer-Aided Design*, FMCAD '08, pages 15:1–15:9, Piscataway, NJ, USA, 2008. IEEE Press.

[14] Temesghen Kahsai and Cesare Tinelli. PKIND: a parallel k-induction based model checker. In *International Workshop on Parallel and Distributed Methods in verifiCation*, pages 55–62, 2011.

[15] Tim King, Clark Barrett, and Bruno Dutertre. Simplex with sum of infeasibilities for SMT. In *Proceedings of the 2013 International Conference on Formal Methods in Computer-Aided Design*, 2013.

[16] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In *ANNUAL ACM IEEE DESIGN AUTOMATION CONFERENCE*, pages 530–535. ACM, 2001.

[17] Greg Nelson and Derek Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems*, 1(2):245–57, 1979.

[18] Duckki Oe, Andrew Reynolds, and Aaron Stump. Fast and flexible proof checking for SMT. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, SMT '09, pages 6–13, New York, NY, USA, 2009. ACM.

[19] Aaron Stump, Clark W. Barrett, and David L. Dill. CVC: A cooperating validity checker. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the* 14^{th} *International Conference on Computer*

Aided Verification (CAV '02), volume 2404 of *Lecture Notes in Computer Science*, pages 500–504. Springer-Verlag, July 2002. Copenhagen, Denmark.

[20] Geoff Sutcliffe. The CADE ATP System Competition. `http://www.cs.miami.edu/~tptp/CASC/24/`.